UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P O Box 1450
Alexandria, Virginia 22313-1450
www uspto gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/646,309	08/22/2003	Gregory M. Wright	SUN-P9042	9198

57960 7590 11/24/2008
PVF -- SUN MICROSYSTEMS INC.
C/O PARK, VAUGHAN & FLEMING LLP
2820 FIFTH STREET
DAVIS, CA 95618-7759

EXAMINER	
CHEN, QING	
ART UNIT	**PAPER NUMBER**
2191	
MAIL DATE	**DELIVERY MODE**
11/24/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/646,309	WRIGHT ET AL.	
	Examiner	Art Unit	
	Qing Chen	2191	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *22 August 2008*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1,2,4-11,13-18 and 28-35* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1,2,4-11,13-18 and 28-35* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)	4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)	5) ☐ Notice of Informal Patent Application
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.	6) ☐ Other: _____.

DETAILED ACTION

1. This Office action is in response to the amendment filed on August 22, 2008.

2. **Claims 1, 2, 4-11, 13-18, and 28-35** are pending.

3. **Claims 1, 10, 28, and 32** have been amended.

4. **Claims 3, 12, 19-27, and 36-39** have been canceled.

Response to Amendment

Claim Objections

5. **Claims 1, 2, 4-11, 13-18, and 28-35** are objected to because of the following informalities:

- **Claims 1, 2, 4-6, 9-11, 13-15, 18, 28, 29, 31-33, and 35** recite the limitation "the native code method." Applicant is advised to change this limitation to read "the selected native code method" for the purpose of providing it with proper explicit antecedent basis.
- **Claims 7 and 8** depend on Claim 1 and, therefore, suffer the same deficiency as Claim 1.
- **Claims 16 and 17** depend on Claim 10 and, therefore, suffer the same deficiency as Claim 10.
- **Claim 30** depends on Claim 28 and, therefore, suffers the same deficiency as Claim 28.
- **Claim 34** depends on Claim 32 and, therefore, suffers the same deficiency as Claim 32.

- **Claims 2, 9, 11, and 18** recite the limitation "the call to the native code method." Applicant is advised to change this limitation to read "the call to any native code method" for the purpose of providing it with proper explicit antecedent basis.
- **Claims 6, 15, 31, and 35** recite the limitation "wherein combining the intermediate representation for the native code method with the intermediate representation associated with the application running on the virtual machine." Applicant is advised to change this limitation to read "wherein integrating the intermediate representation for the selected native code method with the intermediate representation associated with the application running on the virtual machine" for the purpose of providing it with proper explicit antecedent basis.
- **Claims 28, 30, 32, and 34** recite the limitation "the callback by the native code method." Applicant is advised to change this limitation to read "the callback by any native code method" for the purpose of providing it with proper explicit antecedent basis.
- **Claims 29 and 31** depend on Claim 28 and, therefore, suffer the same deficiency as Claim 28.
- **Claims 33 and 35** depend on Claim 32 and, therefore, suffer the same deficiency as Claim 32.

Appropriate correction is required.

Claim Rejections - 35 USC § 112

6. The following is a quotation of the first paragraph of 35 U.S.C. 112:

The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same and shall set forth the best mode contemplated by the inventor of carrying out his invention.

7. **Claims 1, 2, 4-11, 13-18, and 28-35** are rejected under 35 U.S.C. 112, first paragraph, as failing to comply with the written description requirement. The claim(s) contains subject matter which was not described in the specification in such a way as to reasonably convey to one skilled in the relevant art that the inventor(s), at the time the application was filed, had possession of the claimed invention.

Claims 1, 10, 28, and 32 recite the limitation "wherein the optimizing the interactions involves using contextual information from within the integrated intermediate representation that is generated from the native code method as well as the application in order to optimize calls to the native code method by the application within the integrated intermediate representation." The subject matter is not properly described in the application as filed, since the drawings and specification only disclose generating native code from the integrated intermediate representation and optimizing interactions between the application running on the virtual machine and the native code method *(see Figure 2, element 212 and paragraph [0031])*. The specification lacks disclosure on "wherein the optimizing the interactions involves using contextual information from within the integrated intermediate representation that is generated from the native code method as well as the application in order to optimize calls to the native code method by the application within the integrated intermediate representation."

On page 14 of the "Remarks" (received on August 22, 2008), the Applicant submits that support for the amendments may be found in Figure 2 of the drawings and paragraphs [0003],

[0004], and [0028]-[0032] of the specification. However, after a careful review of the drawings and the specification by the Examiner, it appears that there is no support for the amendments in Figure 2 and paragraphs [0003], [0004], and [0028]-[0032] with respect to "wherein the optimizing the interactions involves using contextual information from within the integrated intermediate representation that is generated from the native code method as well as the application in order to optimize calls to the native code method by the application within the integrated intermediate representation." Particularly, it appears that the original disclosure does not provide any support for the claimed feature of "using contextual information from within the integrated intermediate representation." The specification, at best, only discloses setting up a context for decompilation in paragraph [0027] and that additional information can be used to improve the optimization process in paragraph [0032]. No mentioning of "contextual information" can be found in paragraphs [0003], [0004], and [0028]-[0032]. Because the specification does not adequately support the claimed subject matter, it would not reasonably convey to one skilled in the relevant art that the inventor(s), at the time the application was filed, had possession of the claimed invention.

Claims 2 and 4-9 depend on Claim 1 and, therefore, suffer the same deficiency as Claim 1.

Claims 11 and 13-18 depend on Claim 10 and, therefore, suffer the same deficiency as Claim 10.

Claims 29-31 depend on Claim 28 and, therefore, suffer the same deficiency as Claim 28.

Claims 33-35 depend on Claim 32 and, therefore, suffer the same deficiency as Claim 32.

8. The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

9. **Claims 28-35** are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

Claims 28 and 32 recite the limitation "the interactions." There is insufficient antecedent basis for this limitation in the claims. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "the callback" for the purpose of further examination.

Claims 29-31 depend on Claim 28 and, therefore, suffer the same deficiency as Claim 28.

Claims 33-35 depend on Claim 32 and, therefore, suffer the same deficiency as Claim 32.

Claim 32 recites the limitation "the combined intermediate representation." There is insufficient antecedent basis for this limitation in the claim. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "the integrated intermediate representation" for the purpose of further examination.

Claims 33-35 depend on Claim 32 and, therefore, suffer the same deficiency as Claim 32.

Claim Rejections - 35 USC § 103

10. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

11. **Claims 1, 2, 4-7, 10, 11, 13-16, and 28-35** are rejected under 35 U.S.C. 103(a) as being unpatentable over **US 6,289,506 (hereinafter "Kwong")** in view of **US 6,662,358 (hereinafter "Berry")**.

As per **Claim 1**, Kwong discloses:

- selecting a call to any native code method to be optimized within the virtual machine *(see Figure 7: 730; Column 8: 32-35, "... if the programmer decides to try to improve performance, then at step 730, he may select some of the Java program methods on the candidate list from step 720 for optimization.")*;

- decompiling at least part of the selected native code method into an intermediate representation *(see Figure 7: 735; Column 8: 38-45, "... a user may decide to de-compile earlier native compiled code back to bytecode format. The de-compile process may be used for instance when a user determines that the native compiled code does not present the desired performance and the user wants to revert the native compiled code back to Java bytecode. A dynamic linked library (DLL) is created for the compiled native program methods at step 740.")*;

- obtaining an intermediate representation associated with the application running on the virtual machine which interacts with the selected native code method *(see Figure 7: 705; Column 8: 23-25, "A programmer would first write a computer program in the Java programming language in step 705.")*; [Examiner's Note: The source code of the computer program is modified (intermediate representation) after an iteration of the optimization loop by incorporating the DLL for the native methods into the source code of the computer program.]

- integrating the intermediate representation for the selected native code method into the intermediate representation associated with the application running on the virtual machine to form an integrated intermediate representation *(see Figure 7: 705 and 740; Column 10: 8-10, "... the Java application now comprises of Lib.dll 1060, A.class 1010, and B.class 1020 and may* *be executed on a Java VM 1080.")*; [Examiner's Note: Figure 7 clearly illustrates that the DLL for the native methods (intermediate representation for the selected native code method) is incorporated into the modified source code of the computer program (intermediate representation associated with the application) and thus, the modified source code of the computer program now contains the DLL for the native methods (integrated intermediate representation).] and

- generating native code from the integrated intermediate representation, wherein the native code generation process optimizes interactions between the application running on the virtual machine and the selected native code method, wherein optimizing the interactions involves using information from within the integrated intermediate representation that is generated from the selected native code method as well as the application in order to optimize calls to the selected native code method by the application within the integrated intermediate representation *(see Figure 7: 710, 715, and 720; Column 8: 46 and 47, "... a programmer may repeat these steps to further refine and optimize the program."; Column 9: 9-11, "Once the bytecodes are in the Java VM 840, they are interpreted by a Java interpreter 842 or turned into native machine code by the JIT compiler 844.")*.

However, Kwong does not disclose:

- wherein optimizing the interactions involves using contextual information.

Berry discloses:

- wherein optimizing the interactions involves using contextual information *(see Column 3: 17-27, "A method and system for monitoring performance of a program is provided. A trace record containing a call stack associated with the program is periodically generated. An occurrence of a selected event or a timer interrupt is detected, and in response, an execution* *context sample is obtained that contains a process identifier, a thread identifier, a program counter, and a stack pointer. A trace record containing the execution context sample data is generated. During post-processing, the execution context samples are compared with a call stack to identify the routine associated with the execution sample data.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Berry into the teaching of Kwong to include wherein optimizing the interactions involves using contextual information. The modification would be obvious because one of ordinary skill in the art would be motivated to enhance the performance of a software program through software profiling *(see Berry – Column 1: 37-43).*

As per **Claim 2**, the rejection of **Claim 1** is incorporated; and Kwong further discloses:

- wherein selecting the call to any native code method involves selecting the call based upon at least one of: the execution frequency of the call *(see Column 4: 11-19, "An analysis tool may track the Java program methods entered and exited in memory, establish a relationship between parent and child methods called, record every called program method, and time spend in each method. In another embodiment, an analysis tool may keep track of the Java methods being loaded into memory along with active software executing on the system. A tuning tool may determine the most active classes and methods in a Java application and list possible candidates for native compilation.")*; and the overhead involved in performing the call to any native code method as compared against the amount of work performed by the selected native code method.

As per **Claim 4**, the rejection of **Claim 1** is incorporated; and Kwong further discloses:

- wherein optimizing interactions between the application running on the virtual machine and the selected native code method involves optimizing callbacks by the selected native code method into the virtual machine *(see Column 7: 9-12, "In order to maintain the state of the Java VM 430 and make system calls, the compiled Java code 440 may make calls 450 into the Java VM 430.")*.

As per **Claim 5**, the rejection of **Claim 4** is incorporated; however, Kwong does not disclose:

- wherein optimizing callbacks by the selected native code method into the virtual machine involves optimizing callbacks that access heap objects within the virtual machine.

Official Notice is taken that it is old and well-known within the computing art to allow callbacks to access heap objects within the virtual machine. Applicant has submitted in the specification that JNI™ provides an interface through which native code can manipulate heap objects within the JVM™ in a platform-independent way *(see Page 2, Paragraph [0004])*. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to include wherein optimizing callbacks by the selected native code method into the virtual machine involves optimizing callbacks that access heap objects within the virtual machine. The modification would be obvious because one of ordinary skill in the art would be motivated to allow the implementation of a Java™ object to remain transparent to the native code.

As per **Claim 6**, the rejection of **Claim 4** is incorporated; and Kwong further discloses:

- wherein the virtual machine is a platform-independent virtual machine *(see Figure 2: 212)*; and

- wherein integrating the intermediate representation for the selected native code method with the intermediate representation associated with the application running on the virtual machine involves integrating calls provided by an interface for accessing native code into the selected native code method *(see Column 5: 41-44, "A Java Native Interface (JNI) may exist with the Java VM 212. The Java Native Interface is a standard programming interface for writing Java native methods and embedding the Java VM into native applications."; Column 10: 10-13, "When the method in A.class 1010 is native compiled, it needs to use the Java native interface 1070 to access the field b in class B 1020.")*.

As per **Claim 7,** the rejection of **Claim 1** is incorporated; and Kwong further discloses:

- wherein obtaining the intermediate representation associated with the application running on the virtual machine involves recompiling a corresponding portion of the application *(see Column 8: 48-50, "The process of monitoring and compiling bytecode/de-compiling native code may be repeated until the desired performance is obtained.")*.

Claims 10, 11, and 13-16 are computer-readable storage device claims corresponding to the method claims above (Claims 1, 2, and 4-7) and, therefore, are rejected for the same reasons set forth in the rejections of Claims 1, 2, and 4-7.

As per **Claim 28,** Kwong discloses:

- deciding to optimize a callback by any native code method into the virtual machine *(see Figure 7: 730; Column 7: 9-12, "In order to maintain the state of the Java VM 430 and make system calls, the compiled Java code 440 may make calls 450 into the Java VM 430."; Column 8: 32-35, "... if the programmer decides to try to improve performance, then at step 730, he may select some of the Java program methods on the candidate list from step 720 for optimization.")*;

- decompiling at least part of the selected native code method into an intermediate representation *(see Figure 7: 735; Column 8: 38-45, "... a user may decide to de-compile earlier native compiled code back to bytecode format. The de-compile process may be used for instance when a user determines that the native compiled code does not present the desired performance and the user wants to revert the native compiled code back to Java bytecode. A dynamic linked library (DLL) is created for the compiled native program methods at step 740.")*;

- obtaining an intermediate representation associated with the application running on the virtual machine which interacts with the selected native code method *(see Figure 7: 705; Column 8: 23-25, "A programmer would first write a computer program in the Java programming language in step 705.")*; [Examiner's Note: The source code of the computer program is modified (intermediate representation) after an iteration of the optimization loop by incorporating the DLL for the native methods into the source code of the computer program.]

- integrating the intermediate representation for the selected native code method into the intermediate representation associated with the application running on the virtual machine to form an integrated intermediate representation *(see Figure 7: 705 and 740; Column 10: 8-10, "... the Java application now comprises of Lib.dll 1060, A.class 1010, and B.class 1020 and may* *be executed on a Java VM 1080.")*; [Examiner's Note: Figure 7 clearly illustrates that the DLL for the native methods (intermediate representation for the selected native code method) is incorporated into the modified source code of the computer program (intermediate representation associated with the application) and thus, the modified source code of the computer program now contains the DLL for the native methods (integrated intermediate representation).] and

- generating native code from the integrated intermediate representation, wherein the native code generation process optimizes the callback by any selected native code method into the virtual machine, wherein optimizing the callback involves using information from within the integrated intermediate representation that is generated from the selected native code method as well as the application in order to optimize calls to the selected native code method by the application within the integrated intermediate representation *(see Figure 7: 710, 715, and 720; Column 8: 46 and 47, "... a programmer may repeat these steps to further refine and optimize the program."; Column 9: 9-11, "Once the bytecodes are in the Java VM 840, they are interpreted by a Java interpreter 842 or turned into native machine code by the JIT compiler 844.")*.

However, Kwong does not disclose:

- wherein optimizing the callback involves using contextual information.

Berry discloses:

- wherein optimizing the callback involves using contextual information *(see Column 3: 17-27, "A method and system for monitoring performance of a program is provided. A trace record containing a call stack associated with the program is periodically generated. An occurrence of a selected event or a timer interrupt is detected, and in response, an execution* *context sample is obtained that contains a process identifier, a thread identifier, a program counter, and a stack pointer. A trace record containing the execution context sample data is generated. During post-processing, the execution context samples are compared with a call stack to identify the routine associated with the execution sample data.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Berry into the teaching of Kwong to include wherein optimizing the callback involves using contextual information. The modification would be obvious because one of ordinary skill in the art would be motivated to enhance the performance of a software program through software profiling *(see Berry – Column 1: 37-43).*

As per **Claim 29**, the rejection of **Claim 28** is incorporated; and Kwong further discloses:

- wherein the native code generation process also optimizes calls to the selected native code method by the application *(see Column 8: 32-35, "... if the programmer decides to try to improve performance, then at step 730, he may select some of the Java program methods on the candidate list from step 720 for optimization.").*

As per **Claim 30**, the rejection of **Claim 28** is incorporated; however, Kwong does not disclose:

- wherein optimizing the callback by any native code method into the virtual machine involves optimizing a callback that accesses a heap object within the virtual machine.

Official Notice is taken that it is old and well-known within the computing art to allow callbacks to access heap objects within the virtual machine. Applicant has submitted in the specification that JNI™ provides an interface through which native code can manipulate heap objects within the JVM™ in a platform-independent way *(see Page 2, Paragraph [0004])*. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to include wherein optimizing the callback by any native code method into the virtual machine involves optimizing a callback that accesses a heap object within the virtual machine. The modification would be obvious because one of ordinary skill in the art would be motivated to allow the implementation of a Java™ object to remain transparent to the native code.

As per **Claim 31,** the rejection of **Claim 28** is incorporated; and Kwong further discloses:

- wherein the virtual machine is a platform-independent virtual machine *(see Figure 2: 212)*; and

- wherein integrating the intermediate representation for the selected native code method with the intermediate representation associated with the application running on the virtual machine involves integrating calls provided by an interface for accessing native code into the selected native code method *(see Column 5: 41-44, "A Java Native Interface (JNI) may exist with the Java VM 212. The Java Native Interface is a standard programming interface for writing Java native methods and embedding the Java VM into native applications."; Column 10: 10-13, "When the method in A.class 1010 is native compiled, it needs to use the Java native interface 1070 to access the field b in class B 1020.")*.

Claims 32-35 are computer-readable storage device claims corresponding to the method claims above (Claims 28-31) and, therefore, are rejected for the same reasons set forth in the rejections of Claims 28-31.

12. **Claims 8 and 17** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Kwong** in view of **Berry** as applied to Claims 1 and 10 above, and further in view of **US 5,491,821 (hereinafter "Kilis")**.

As per **Claim 8**, the rejection of **Claim 1** is incorporated; however, Kwong and Berry do not disclose:

- wherein obtaining the intermediate representation associated the application running on the virtual machine involves accessing a previously generated intermediate representation associated with the application running on the virtual machine.

Kilis discloses:

- wherein obtaining the intermediate representation associated the application running on the virtual machine involves accessing a previously generated intermediate representation associated with the application running on the virtual machine *(see Column 2: 2-4, "If the selected changed facet affects the object itself, then the previous intermediate representation of the object is modified.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Kilis into the teaching of Kwong to include wherein obtaining the intermediate representation associated the application running on the virtual machine involves accessing a previously generated intermediate representation associated with the application running on the virtual machine. The modification would be obvious because one of ordinary skill in the art would be motivated to not reprocess existing information *(see Kilis – Column 1: 40-43).*

Claim 17 is rejected for the same reason set forth in the rejection of Claim 8.

13. **Claims 9 and 18** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Kwong** in view of **Berry** as applied to Claims 1 and 10 above, and further in view of **US 5,805,899 (hereinafter "Evans")**.

As per **Claim 9**, the rejection of **Claim 1** is incorporated; and Kwong further discloses:

- determining a signature of the call to the native code method *(see Column 4: 11-19, "An analysis tool may track the Java program methods entered and exited in memory, establish a relationship between parent and child methods called, record every called program method, and time spend in each method. In another embodiment, an analysis tool may keep track of the Java methods being loaded into memory along with active software executing on the system. A tuning tool may determine the most active classes and methods in a Java application and list possible candidates for native compilation.").*

However, Kwong and Berry do not disclose:

- determining a mapping from arguments of the call to corresponding locations in a native application binary interface (ABI).

Evans discloses:

- determining a mapping from arguments of the call to corresponding locations in a native application binary interface (ABI) *(see Column 7: 29-33, "Shared object 114 provides global symbols to which other objects, such as dynamic executable 120, can bind at runtime. These global symbols are specified in mapfile 130 and describe an Application Binary Interface (ABI) of shared object 114.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Evans into the teaching of Kwong to include wherein determining a mapping from arguments of the call to corresponding locations in a native application binary interface (ABI). The modification would be obvious because one of ordinary skill in the art would be motivated to describe the low-level interface between an application program and the operating system, its libraries, or components of the application program.

Claim 18 is rejected for the same reason set forth in the rejection of Claim 9.

Response to Arguments

14. Applicant's arguments with respect to Claims 1, 10, 28, and 32 have been considered but are moot in view of the new ground(s) of rejection.

In the Remarks, Applicant argues:

a) Applicant respectfully emphasizes that embodiments of the present invention combine the equivalent elements into a single intermediate representation and optimize the combined single intermediate representation. This is beneficial because embodiments of the present invention generate a combined IR that provides additional information to the optimization process that would not be available in the Kwong system.

Consider, for example, a case where a variable, *var*, is set within a native code method from a generic native code library, and execution of a program segment in the application code depends upon the value of this variable, *var*. In embodiments of the present invention, it is possible to optimize the application code based on the value of *var* in the native code, since they are both combined into a single IR and available as one entity to the optimization process. This is not possible in the system of Kwong involving calls to any native code method from a Java application.

There is nothing in Kwong, either explicit or implicit, that discloses generating intermediate representations for both the application program as well as any native code methods, integrating both the intermediate representations, and performing an optimization on this integrated intermediate representation. Hence, it is not possible to use the system of Kwong to improve the optimization process by using additional information from the combined IR using both the application program IR as well as the IR for any native code.

Examiner's response:

a) Examiner disagrees. Applicant's arguments are not persuasive for at least the following reasons:

First, without acquiescing to the Applicant's assertion that there is nothing in Kwong, either explicit or implicit, that discloses generating intermediate representations for both the application program as well as any native code methods, the Examiner first submits that the plain language of the claims does not require generating intermediate representations for both the application program as well as any native code methods. The claims recite, exactly, "decompiling at least part of the selected native code method into an intermediate representation" and "obtaining an intermediate representation associated with the application running on the virtual machine which interacts with the native code method." Thus, the claims are only limited to the scope of decompiling the selected native code method into an intermediate representation and obtaining an intermediate representation associated with the application, respectively. The claims are not limited to the scope of "generating" the intermediate representations. Applicant is reminded that in order for such limitations to be considered, the claim language requires to specifically recite such limitations in the claims, otherwise broadest reasonable interpretations of the broadly claimed limitations are deemed to be proper.

Second, with respect to the Applicant's assertion that there is nothing in Kwong, either explicit or implicit, that discloses generating an intermediate representation for any native code methods, as previously pointed out in the Non-Final Rejection (mailed on 05/22/2008) and further clarified hereinabove in the § 103(a) rejection and hereinafter, the Examiner respectfully submits that Kwong clearly discloses "decompiling at least part of the selected native code method into an intermediate representation" *(see Figure 7: 735 and 740; Column 8: 38-45, "... a user may decide to de-compile earlier native compiled code back to bytecode format. The de-compile process may be used for instance when a user determines that the native compiled code does not present the desired performance and the user wants to revert the native compiled code back to Java bytecode. A dynamic linked library (DLL) is created for the compiled native* *program methods at step 740.")*. Note that a user may de-compile the selected native methods back to bytecode. After the user is satisfied with the performance of the selected native methods, they are compiled back into native processor code. Subsequently, a DLL (intermediate representation) is created for the native methods.

Third, with respect to the Applicant's assertion that there is nothing in Kwong, either explicit or implicit, that discloses generating an intermediate representations for the application program, as previously pointed out in the Non-Final Rejection (mailed on 05/22/2008) and further clarified hereinabove in the § 103(a) rejection and hereinafter, the Examiner respectfully submits that Kwong clearly discloses "obtaining an intermediate representation associated with the application running on the virtual machine which interacts with the selected native code method" *(see Figure 7: 705; Column 8: 23-25, "A programmer would first write a computer program in the Java programming language in step 705.")*. As clarified hereinabove in the § 103(a) rejection, the source code of the computer program is modified (intermediate representation) after an iteration of the optimization loop by incorporating the DLL for the native methods into the source code of the computer program. Thus, one of ordinary skill in the art would readily comprehend that, as illustrated in Figure 7, by adding the DLL code into the source code of the computer program at the end of the optimization loop creates a modified (and improved) version of the computer program that can be further tested/debugged until the user is satisfied with its performance.

Fourth, Examiner would like to point out that for at least the definiteness of the limitation of "intermediate representation" recited in the claims was analyzed in accordance with the level of ordinary skill in the art and in light of the specification. The specification provides an exemplary definition for "intermediate representation" as to "include any intermediate representation of code between the original source code and the final binary executable code for the application. For example, the intermediate representation can include, <u>but is not limited to</u>, modified source code, platform-independent byte codes, assembly code, an intermediate representation for computational operations used within a compiler, or binary code that is not in final executable form (emphasis added)" *(see paragraph [0028])*. Such description provided by the Applicant is, at most, an exemplary instance of the term rather than an *explicit and deliberate definition* for the term. Accordingly, the scope of "intermediate representation" is not limited to its ordinary and customary meaning as understood by those of ordinary skill in the art. Thus, as the claims are interpreted as broadly as their terms reasonably allow (see MPEP § 2111.01(I)), the interpretation of a broad limitation of "intermediate representation" as a DLL file and modified source code and the like by one of ordinary skill in the art is considered to be reasonable by its plain meaning and/or according to its exemplary definition as set forth in the specification.

Fifth, with respect to the Applicant's assertion that there is nothing in Kwong, either explicit or implicit, that discloses integrating both the intermediate representations, as previously pointed out in the Non-Final Rejection (mailed on 05/22/2008) and further clarified hereinabove in the § 103(a) rejection and hereinafter, the Examiner respectfully submits that Kwong clearly discloses "integrating the intermediate representation for the selected native code method into the intermediate representation associated with the application running on the virtual machine to form an integrated intermediate representation" *(see Figure 7: 705 and 740; Column 10: 8-10, "... the Java application now comprises of Lib.dll 1060, A.class 1010, and B.class 1020 and may* *be executed on a Java VM 1080.").* As clarified hereinabove in the § 103(a) rejection, Figure 7 clearly illustrates that at the end of the optimization loop, the DLL for the native methods (intermediate representation for the selected native code method) is incorporated into the modified source code of the computer program (intermediate representation associated with the application) and thus, the modified source code of the computer program now contains the DLL for the native methods (integrated intermediate representation).

Sixth, with respect to the Applicant's assertion that there is nothing in Kwong, either explicit or implicit, that discloses performing an optimization on this integrated intermediate representation, as previously pointed out in the Non-Final Rejection (mailed on 05/22/2008) and further clarified hereinafter, the Examiner respectfully submits that Kwong clearly discloses "generating native code from the integrated intermediate representation, wherein the native code generation process optimizes interactions between the application running on the virtual machine and the native code method" *(see Figure 7: 710, 715, and 720; Column 8: 46 and 47, "... a programmer may repeat these steps to further refine and optimize the program."; Column 9: 9-11, "Once the bytecodes are in the Java VM 840, they are interpreted by a Java interpreter 842 or turned into native machine code by the JIT compiler 844.").* Note that, as clarified hereinabove, the modified source code of the computer program containing the DLL for the native methods (integrated intermediate representation) is further compiled (generating native code) and tested/debugged until the desired performance is obtained.

Therefore, for at least the reasons set forth above, the rejections made under 35 U.S.C. § 103(a) with respect to Claims 1, 10, 28, and 32 are proper and therefore, maintained.

Conclusion

15. The prior art made of record and not relied upon is considered pertinent to Applicant's disclosure.

16. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

17. Any inquiry concerning this communication or earlier communications from the Examiner should be directed to Qing Chen whose telephone number is 571-270-1071. The Examiner can normally be reached on Monday through Thursday from 7:30 AM to 4:00 PM. The Examiner can also be reached on alternate Fridays.

If attempts to reach the Examiner by telephone are unsuccessful, the Examiner's supervisor, Wei Zhen, can be reached on 571-272-3708. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the TC 2100 Group receptionist whose telephone number is 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/Q. C./

Examiner, Art Unit 2191

/Wei Y Zhen/

Supervisory Patent Examiner, Art Unit 2191